ITNB – Technical Assessment

Stage 2: Written Design Questions

Question 1: Role-Based Access Control (RBAC) Implementation

To implement strict RBAC within a GroundX-powered system for an enterprise environment like SharePoint, I would design a hybrid approach combining Bucket Isolation for hard boundaries and Metadata Filtering for granular permissions.

1. **Document-User Association**

We need to map the enterprise's existing identity provider roles to GroundX entities.

Using buckets: I would create separate GroundX buckets for top-level security domains that should never cross-pollinate for example legal-sensitive, hr-confidential, public-knowledge. This ensures that even a software bug in the query layer cannot leak strictly confidential data, as the API key or bucket ID itself acts as a hard boundary.

Using metadata tags: Within shared buckets for example engineering-docs, I would utilize the ‘metadata’ field during ingestion. Every document would be tagged with an access control list derived from the source system.

ExampleMetadata:
{"allowed_roles": ["manager", "editor"], "department": "sales", "owner_id": "u123"}

2. Query-Time Enforcement

Security must be enforced at the application middleware layer never by the client.

a) Authentication: The user authenticates via SSO (OIDC/SAML), and the Orchestrator resolves their claims for example roles=[‘intern’], dept=’sales’.
b) Query Rewriting: When the user submits a query, the Orchestrator intercepts it. It selects the appropriate buckets based on the user's department. It appends a metadata filter to the GroundX search request.

Logic: search(query=”...”, filter={“department”: “sales”, “allowed_roles”: {“$in”: [“intern”, “all”]}})

c) As a final defense in depth, the application verifies the metadata of returned chunks before displaying them, ensuring no result slips through due to index latency or filter edge cases.

3. Limitations & Considerations

→ If user permissions change in active directory, there is a lag before the document metadata in GroundX can be updated so re-ingestion is required). We would need a "permissions sync" pipeline.

→ Complex hierarchical permissions are difficult to flatten into simple key-value metadata tags.

→ Metadata filtering is soft security. If the developer forgets to apply the filter in the code, data leaks. Bucket isolation is hard security and generally safer for high-compliance data.

Question 2: Scaling RAG for Large and Dynamic Knowledge Bases

Scaling RAG for corporate clients with thousands of volatile documents requires moving from a script-based approach to an Event-Driven Architecture. The goal is to minimize the time-to-knowledge where the latency between a document update and its availability in the chat.

1. Handling Large, Frequent Changes

Crawling every night is insufficient for dynamic enterprise data. I would implement:

→ Integrate directly with the source systems like SharePoint Webhooks, Database transaction logs. When a user saves a file, a webhook fires.

→ These events act as producers for a message queue for example Kafka. This decouples the bursting nature of document updates from the ingestion throughput.

→ A pool of consumers pulls tasks from the queue to process, chunk, and upsert documents into GroundX asynchronously. This ensures the chat interface remains responsive even during massive data dumps.

→ We must handle deletions as robustly as additions. The event for "File Deleted" must immediately trigger a (delete_documents) call to GroundX to prevent the AI from hallucinating based on obsolete policies.

2. User empowerment Human-in-the-Loop

I would absolutely empower users to manage their own documents via a Knowledge dashboard.

→ Users often distrust black box scrapers. Showing them, exactly what files are indexed and their status like processing, ready, error etc builds confidence.

→ Users should be able to manually boost or archive documents. An old project plan might technically be relevant but practically useless, the user knows this, the AI doesn't.

→ If a user just uploaded a critical policy, they should have a "Sync Now" button to bypass the standard queue priority, giving them a sense of control.

3. Workflows for efficiency and upto date

To keep retrieval efficient, I would deploy specialized background agents:

→ Periodically scan the vector store against the source system to identify duplicate documents (files that exist in the index but were silently deleted in the source).

→ An LLM-based worker that runs during ingestion to generate better metadata summaries and suggested questions. It enriches the raw text with semantic tags, improving retrieval accuracy beyond simple vector similarity.

→ If a user downvotes an answer in the chat, the system captures the query and the retrieved chunks. This data feeds into an automated evaluation pipeline to flag underperforming documents for review or re-chunking.

Prompts used:

Here is a record of how I utilized AI prompts to accelerate development. I used them primarily for boilerplate generation, regex pattern matching, and library-specific syntax lookups to focus my time on the core logic.

Regex for Content Cleaning

Context: I needed a robust way to strip cookie banners and excess whitespace from the scraped markdown without breaking the structure.

Prompt:

Write a python regex pattern to identify and remove common cookie consent banners (like 'Manage Cookies', 'Allow All') from a text string. Also include a pattern to collapse multiple newlines into a single paragraph break.

Rich CLI Boilerplate

Context: I wanted a polished terminal interface but didn't want to spend an hour reading rich docs for specific panel and table configurations.

Prompt:

Show me a snippet using the Python ‘rich’ library to creating a console application with:

1. A colored header panel.
2. A spinner for async tasks.
3. A table for displaying search results with columns for 'Title' and 'Source URL'.

Keep the styling minimal and professional."

GroundX API Integration

Context: I needed to quickly map the GroundX SDK retrieval response to my internal data structure.

Prompt:

I'm using the ‘groundx’ python SDK. When calling client.search.content(), what is the exact structure of the response object? Write a helper function to parse the response and extract the ‘text’, ‘source_url’, and ‘score’ into a list of dictionaries, handling cases where metadata might be missing.

Pydantic Configuration

Context: Setting up type-safe environment variable loading.

Prompt:

Generate a ‘pydantic-settings’ ‘BaseSettings’ class that loads ‘GROUNDX_API_KEY’ and ‘OPENAI_API_KEY’ from a ‘.env’ file. Include validation to ensure the api keys are not empty strings.